ECE598: Advanced Operating Systems – Homework 8

Spring 2015

Locking and Context Switch

Due: Thursday, 30 April 2015, 9:30am

Answer the following multiple choice questions.

1. In the following memory allocation code, what type of allocator is used?

   (a) First fit

   (b) Next fit

   (c) Best fit

   (d) Worst fit

```
static int find_free(int num_chunks) {

    int i,j;

    for(i=0;i<max_chunk;i++) {
        if (!memory_test_used(i)) {
            for(j=0;j<num_chunks;j++) {
                if (memory_test_used(i+j)) break;
            }
            if (j==num_chunks) {
                return i;
            }
        }
    }

    return -1;
}

void *memory_allocate(int size) {

    int first_chunk,num_chunks,i;

    if (size==0) size=1;

    num_chunks = ((size-1)/CHUNK_SIZE)+1;

    first_chunk=find_free(num_chunks);

    if (first_chunk<0) {
        printk("Error!  Could not allocate %d of memory!\n",size);
        return NULL;
    }

    for(i=0;i<num_chunks;i++) {
        memory_mark_used(first_chunk+i);
    }

    return (void *)(first_chunk*CHUNK_SIZE);
}
```

2. In the following code, what kind of scheduling alogrithm is implemented?

   (a) Round-robin

   (b) Priority-based

   (c) Real-time

   (d) Completely fair

```
i=current_process;

save_process(i,pcb);

while(1) {
    i++;
    if (i>=MAX_PROCESSES) i=0;
    if ((process[i].valid) && (process[i].ready)) break;
}

/* switch to new process */
run_process(i,pcb[17]);
```

3. In the following context switch code (which happens in IRQ mode), why is the `sp` register updated?

   (a) To set up the userspace stack.

   (b) To avoid leaking memory on the stack every context switch.

   (c) There is no need to update it, the code is wrong.

```
asm volatile(
    "mov r0, %[our_sp]\n"
    "msr SPSR_cxsf, %[our_spsr]\n"
    "mov lr, %[return_pc]\n"
    "mov sp,%[irq_stack]\n"
    "ldmia r0, {r0 - lr}^\n"        /* the ^ means load user regs */
    "nop\n"
    "movs pc, lr\n"                 /* movs with pc changes mode */
    : /* output */
    :       [our_sp] "r"(our_sp),
        [return_pc] "r"(return_pc),
        [irq_stack] "r"(irq_stack),
        [our_spsr] "r"(our_spsr) /* input */
    : "lr", "sp", "r0", "memory" /* clobbers */ );
```

4. In the following code for loading a process into the process table, what is the earliest location you can unlock without causing a potential race condition?

   (a) A

   (b) B

   (c) C

   (d) D

   (e) E

   (f) B and C

```
    mutex_lock(&load_process_lock);

    /* Find free process */
    for(i=0;i<MAX_PROCESSES;i++) {
        if (!process[i].valid) break;
    }

/* A */
    if (i==MAX_PROCESSES) {
        printk("ERROR: No free process slot!\n");
/* B */
        return -1;
    }

    process[i].valid=1;

/* C */
    /* Allocate Memory */
    binary_start=(char *)memory_allocate(size);
    stack_start=(char *)memory_allocate(stack_size);

/* D */
    /* Load executable */
    memcpy(binary_start,data,size);

    /* Set name */
    strncpy(process[i].name,name,32);

    /* Set up initial conditions */
    process[i].running=0;
    process[i].ready=0;
    process[i].time=0;
/* E */
```

Submit your work

E-mail the file containing your answers to the questions to me by the homework deadline.